PATENT ABSTRACTS OF JAPAN

(11)Publication number : 09-152961
(43)Date of publication of application : 10.06.1997

(51)Int.Cl. G06F 9/06
G06F 9/445

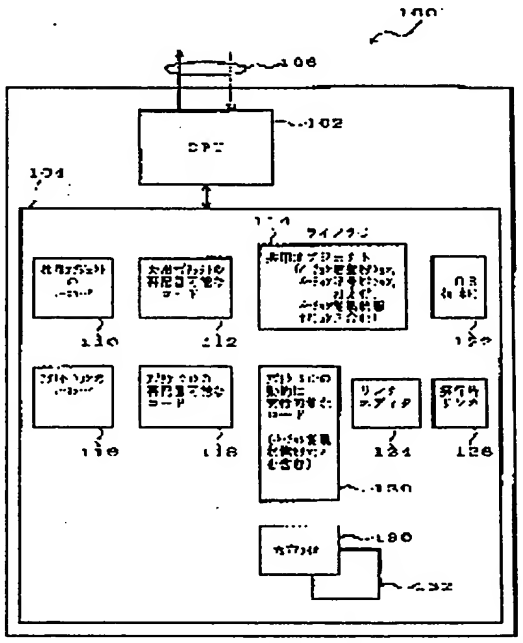
(21)Application number : 08-195380
(22)Date of filing : 05.07.1996

(71)Applicant : SUN MICROSYST INC
(72)Inventor : EVANS RODRICK I
GINGELL ROBERT A

(30)Priority
Priority number : 95 499062 Priority date : 06.07.1995 Priority country : US

(54) METHOD FOR IMPARTING VERSION SETTING INFORMATION TO SOFTWARE PROGRAM AND DEVICE THEREFOR

(57)Abstract:
PROBLEM TO BE SOLVED: To enable an easy and efficient version setting by confirming whether the object for which a version setting is performed is possible to be utilized as an execution time linker.
SOLUTION: At the time of an execution, an execution time linker 126 executes an object 120 which is possible to be dynamically executed by preparing a processing file 122 and executing the file when all the versions of a shared object 114 that the object 120 requires. Thus, a link editor 124 decides which version of the shared object is required for an application program. The execution time linker 26 maps the objects of the memory and couples the objects. Thus, the execution time linker 126 couples the objects just as the linker 26 is instructed by the link editor 124. Further, when a right version does not exist, the execution time linker 126 generates an error.



LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]

[Date of extinction of right]

* NOTICES *

1.This document has been translated by computer. So the translation may not reflect the original precisely.
2.**** shows the word which can not be translated.
3.In the drawings, any words are not translated.

CLAIMS

[Claim(s)]
[Claim 1] The step which is the approach of giving version setting information to a software program, and prepares the object code for a software program, The step which prepares the mapfile which shows the version name about the version of said software program, The information which shows the version name of said software program so that it may be added to said object code according to said mapfile The approach by which said object code is linked, the step which generates by this the object by which a version setup was carried out is provided, and these steps are performed by data processing system.
[Claim 2] The step which is the approach of giving version setting information to a software program, and prepares the 1st object code for the 1st software program, The step which prepares the mapfile which shows the version name about the version of said 1st software program, The information which shows the version name of said 1st software program so that it may be added to said 1st object code according to said mapfile The step which links said 1st object code and generates by this the object by which a version setup was carried out, The step which prepares the 2nd object code for the 2nd software program, It is the step which links said 2nd object code to said object by which a version setup was carried out. The step this step judges the version of said 1st software program needed for said 2nd software program to be, The information which shows said version needed for said 2nd software program is added to said 2nd object code. By this The approach by which what contains further the step which generates the program which can be performed dynamically is provided, and these steps are performed by data processing system.
[Claim 3] The storage which is equipment which gives version setting information to a software program, and stores the 1st object code for the 1st software program, The storage which stores the mapfile which specifies the version name about the version of said 1st software program, According to said mapfile, the additional information which defines the version name of said version of said 1st software program is attached to said 1st object code. By this The linker which generates the object by which a version setup was carried out, and the storage which stores the 2nd object code for the 2nd software program, By giving the additional information which shows the version needed for said 2nd software program to said 2nd object code Equipment possessing the linker which links said 2nd object code to said object by which a version setup was carried out, and generates by this the program which can be performed dynamically.

[Translation done.]

* NOTICES *

1.This document has been translated by computer. So the translation may not reflect the original precisely.
2.**** shows the word which can not be translated.
3.In the drawings, any words are not translated.

DETAILED DESCRIPTION

[Detailed Description of the Invention]
[0001]
[Field of the Invention] This invention relates to the approach and equipment which offer the dynamic link system which manages modification of the version which the software program followed especially about the approach and equipment which link a software program.
[0002]
[Description of the Prior Art] A thing called a software development can be continuously called process in a present progressive. Although the version of the beginning of software will probably be enough for a task when this software is written, the time passes and it needs an upgrade in connection with the new description being added. When a software application is combined with a common object (it is also called a "library") (binding head), especially this software development process is accompanied by the problem. When a common object is updated or changed, the interface over this common object is also often updated or changed. Furthermore, even when the interface over a common object is not changed, change often arises in a part of function performed by said common object.
[0003]
[Problem(s) to be Solved by the Invention] The thing with the conventional system links an application software program to a common object dynamically at the time of activation (run time). In such a system, it is necessary to check carefully the application software program which accesses this common object whenever the new version of a common object is released. Moreover, also in order to judge whether actuation of said common object has changed (even if the interface over said common object is the same), it is necessary to check said application software program. Conventionally, such a check was performed by the help. The error produced by the mismatching of an interface is discovered only in application software program execution, when the mismatching of an interface is found or a common object does not often perform the same actuation even as it. In such a case, they are a means to judge to which version of a common object a specific application software program tends to link the thing which is the need, and a means to confirm whether the required version of said common object exists during the dynamic link processing at the time of activation.
[0004] Conventionally, the file name of said common object itself was updated for every new version. For this reason, only the highest version of said common object will exist during link processing, and this highest version will have a completely different file name from the old version of this object. Whenever the new version of an object is created about this point, it is desirable to make it not change the identifier of this object. The conventional system is often going to avoid the version check by releasing an application program and a common object (library) as "1 A lump's system." That is, it is not concerned with whether modification of the version of a common object was made, but new application is shipped with all the objects that it needs. It is desirable to enable it to upgrade only a part required in order to cope with version modification in said system about this point.
[0005] It aims at offering the approach and equipment which enable an easy and efficient version setup and which give version setting information to a software program by enabling it to upgrade only a part required whenever it was made in view of the above-mentioned point and the new version of an object is created, in order that this invention can be prevented

from changing the identifier of this object and may cope with version modification.
[0006]
[Means for Solving the Problem] This invention offers the approach and equipment which link a software program dynamically, and offers the version system which grasps modification in the version which the object accessed by the application software program followed. This invention checks the interface used by the application software program, in order to access the object by which a version setup was carried out, and it detects the attempt which accesses the invalid version of said object by said application software program by which a version setup was carried out. In this way, this invention enables evolution by which the object was controlled, maintaining the back compatibility between an application software program and an object.
[0007] This invention offers the version setting system which can be made to carry out labeling of the modification on a notation interface and activation in one object. A linkage editor adds the data (a version definitions section and version notation section) which define all the available versions of this object to the object by which a version setup was carried out at the time of construction. Moreover, a linkage editor adds the data (version subordination condition section) which define the requirements for a version of this application to a software application at the time of construction. An execution-time linker checks whether said object which is needed at whether to be in agreement with the version definition by which the requirements for said software application were stored in said object itself, and said software application and by which a version setup was carried out is available to said execution-time linker at the time of activation.
[0008] A version is the identifier or label recorded on the object. A version may be related with one or two global symbols or more (global symbol), and a version defines a notation interface in this case. Although, as for a version, modification on activation, i.e., modification of the function of an object, only exists when that is not right, the definition of a new global symbol may be an indicator in which modification which is not made is shown. In the case of the latter, the version is called a version "weak (week)." The global symbol and version name on the gestalt and relevant to each version of implementation of this invention explained here are defined as the "mapfile" generated by people. In order to create the object by which a version setup was carried out, said mapfile is inputted into said linkage editor with one or two relocatable (compiled) objects or more at the time of construction. When the link edit of the application is carried out to the object by which a version (it has version setting information) setup was carried out at the time of construction, the subordination condition over the version which contains with a default the global symbol referred to by said application is set as said application. Furthermore, the "weak" subordination condition over a weak definition is set up.
[0009] This invention enables succession of the version definition in one object. A version inherits a version, and thereby, in order to create the interface definition relevant to mutual, it can combine two or more sets of notations. For example, one new version can inherit all the global symbols of an old version. This invention can control the visibility of the version of the object at the time of link edit, and can control an available interface to said application program in practice. Moreover, this invention can also force that said application program requires the weak version of an object.
[0010] The step which this invention is the approach of giving version setting information to a software program, and prepares the object code for a software program in order to attain the above-mentioned purpose, The step which prepares the mapfile which shows the version name about the version of said software program, The information which shows the version name of said software program so that it may be added to said object code according to said mapfile Said object code is linked, the step which generates by this the object by which a version setup was carried out is provided, and these steps are performed by data processing system. Thereby, according to this invention, the object by which a version setup was carried out is automatically generable.
[0011] Furthermore, the step which this invention is the approach of giving version setting

information to a software program, and prepares the 1st object code for the 1st software program, The step which prepares the mapfile which shows the version name about the version of said 1st software program, The information which shows the version name of said 1st software program so that it may be added to said 1st object code according to said mapfile The step which links said 1st object code and generates by this the object by which a version setup was carried out, The step which prepares the 2nd object code for the 2nd software program, It is the step which links said 2nd object code to said object by which a version setup was carried out. The step this step judges the version of said 1st software program needed for said 2nd software program to be, The information which shows said version needed for said 2nd software program is added to said 2nd object code. By this What contains further the step which generates the program which can be performed dynamically is provided, and these steps are performed by data processing system. While the object by which a version setup was carried out is automatically generable by this according to this invention, it can link to this object by which a version setup was carried out dynamically, the 2nd software program, i.e., application software program, and activation of it can be enabled.

[0012] The storage which this invention is equipment which gives version setting information to a software program, and stores the 1st object code for the 1st software program if another viewpoint is followed, The storage which stores the mapfile which specifies the version name about the version of said 1st software program, According to said mapfile, the additional information which defines the version name of said version of said 1st software program is attached to said 1st object code. By this The linker which generates the object by which a version setup was carried out, and the storage which stores the 2nd object code for the 2nd software program, By giving the additional information which shows the version needed for said 2nd software program to said 2nd object code Said 2nd object code is linked to said object by which a version setup was carried out, and the linker which generates by this the program which can be performed dynamically is provided.

[0013] If another viewpoint is followed, in order for the version of the object by which this invention is needed for the object which can be performed dynamically to make it judge to exist during activation of this executable object, It is the computer program product equipped with the usable medium by computer which stored the code which can be read by computer. The 1st program code equipment which makes the 1st object code for the 1st software program prepare for a computer and which can be computer read, The 2nd program code equipment the mapfile which specifies the version name relevant to the version of said 1st software program as a computer is made to prepare for and which can be computer read, According to said mapfile, the information which shows the version name of said 1st software program so that it may be added to said 1st object code Said 1st object code is made to link to a computer, and the 3rd program code equipment which makes by this the object by which a version setup was carried out generate and which can be computer read is provided.

[0014]

[Embodiment of the Invention] Hereafter, the gestalt of 1 implementation of this invention is explained to a detail with reference to an accompanying drawing.

1. Introduction drawing 1 is the block diagram of the computer system 100 concerning this invention. This computer system 100 is equipped with CPU102, memory 104, and I/O line 106. Said computer system 100 may contain many elements of others which are not being illustrated for precision here for a disk drive, a keyboard, a display unit, the network connection section, additional memory, additional CPU, etc. so that I may be understood by this contractor. Said memory 104 stores the library (it is also called a common object) 114, and the source code 110 and the relocatable (compiled) code 112 of this common object 114. Furthermore, said memory 104 stores the source code 116 of an application software program, the relocatable (compiled) code 118 of application 116, and the code (linked) 120 of application 116 that can be performed dynamically. Moreover, said memory 104 stores the linker 126 with the OS (kernel) software 122, the mapfile 130,132, and the linkage editor 124 at the time of activation (run time). Each of said common object 114 and an executable object 120 is created by said linkage editor 124. Each of said common object 114 and an executable

code 120 has the same ELF as Prentice Hall and Executable Linking Format (it is also called "an ELF format" the link format which can be performed, and the following) defined as the 3rd edition of "System V Application Binary Interface" published by Inc.
[0015] However, in this invention, the ELF format of said object 114,120 is extended so that additional data may be included, so that it may explain below. The common object 114 has the format shown in drawing 9 . Said object 120 which can be performed dynamically has the format shown in drawing 14 . Said common object 114 contains the version setting information (version definitions section) for every version of this common object 114, and the public symbol list (version notation section) for every version so that it may explain below. Furthermore, said common object 114 may include the information (version subordination condition section) about the subordination condition (relation) of a version. The executable object 120 includes the information (version subordination condition section) about the subordination condition (relation) of a version. Although not illustrated for precision, said memory 104 also stores the information on others, such as an application program, an operating system, and data, so that I may be understood by this contractor.
[0016] The gestalt of desirable implementation of this invention is carried out under the version 2.5 of a Solaris operating system. Solaris is Sun Microsystems and Inc. It is a trademark. Moreover, Unix is X/OPEN and Ltd. It is the trademark to which it was licensed exclusively and which was registered in the U.S. and other countries. Drawing 2 is drawing showing I/O of the linkage editor 124 of drawing 1 , and shows the creation of a common object by which a version setup was carried out. Although a version setup of a common object is explained below, this invention is usable also in order to perform a version setup of the object which can be performed dynamically, and a relocatable object. Thus, these kinds of objects can include a version definitions section and a version notation section. Said linker editor 124 generates the output of reception and the common object 114 for the input of a mapfile 130 and the relocatable object code 112 for common objects. Said mapfile 130 specifies a global-area (global) notation and a version name for every version of said common object. In the gestalt of the operation explained here, preferably, said mapfile 130 has a format of drawing 8 , and is created by people. In the gestalt of other operations, said mapfile 130 may be created by the compile system.
[0017] Drawing 3 is drawing showing other I/O of the linkage editor 124 of drawing 1 , and shows link editing (edit) of the application program by the common object of drawing 2 . Said linkage editor 124 inputs the common object 114 and the relocatable object 118, and generates the object 120 which can be performed dynamically by processing these objects 114,118. As shown in drawing 4 , in order to show whether the version of said object 114 throat is permitted by this link procedure, a mapfile 132 may be used for this step. Preferably, this mapfile 132 has a format of drawing 13 , and is created by people.
[0018] As shown in drawing 5 , all the versions of the common object 114 for which the object 120 which can be performed dynamically needs the execution-time linker 126 check whether it exists or not at the time of activation. When said all versions exist, said execution-time linker 126 performs the object 120 which can be performed dynamically by creating and performing the processing file 122. In this way, said linkage editor 124 builds the common object by which a version setup was carried out from the relocatable object, and judges which version of said common object is needed for an application program. Said execution-time linker 126 carries out the map of the object of the memory, and joins together. In this way, said execution-time linker 126 only combines an object so that it may be directed by said linkage editor 124. Furthermore, said execution-time linker 126 performs the check check for guaranteeing that the version of the common object 114 which the object 120 which can be performed dynamically needs exists. When a right version does not exist, said execution-time linker 126 generates an error.
[0019] Next, possible modification is explained among various versions of the common object 114. Generally, these modification can be classified into two groups of compatible updating and incompatible updating. Updating with said compatibility is additional updating, i.e., updating by which the available global symbol is maintained till then as it is with the interface over the

common object 114. An example of compatible updating is addition of a global symbol. Since the notation is not removed from an earlier version, the application software by which interface connection was made with the earlier version still operates correctly. Updating without said compatibility changes said existing interface into the condition that the existing application which used the specific existing interface breaks down, or unjust actuation is carried out. As an example of updating without such compatibility, there is size of the argument to addition of the argument to removal of a notation and a function, removal of the argument from a function, and a function or modification of the contents. The error correction to the common object 114 is updating with the compatibility over the existing interface, or is incompatible updating. For example, a compatible error correction changes the internal function of the common object 114, only maintaining the interface defined till then. On the other hand, an incompatible error correction needs modification of the interface over the common object 114.

[0020] 2. The creation above-mentioned paragraph of the version setting information at the time of construction was overall explanation of the processing performed at the time of construction and activation, in order to perform a version setup according to this invention. The following paragraphs explain how said linkage editor 124 adds version setting information to a common object and an application program.

[0021] a. In the gestalt of desirable operation, said linkage editor 124 controls a version setup of a common object according to the version directions information on a mapfile 130 to explain below to the version definition creation about the object by which a version setup was carried out at the time of construction. Although being created by people is desirable as for said mapfile 130, it may be created by software like a compile system. Drawing 6 and drawing 7 are flow charts which show the step performed by the linkage editor 124 of drawing 2 . These steps are a part of steps performed in order to create an object like the common object 114 by which a version setup was carried out. The step of drawing 6 and drawing 7 (and drawing 12) is stored in memory 104, and is performed by CPU102 of drawing 1 which executes the instruction of the linkage editor 124 which used the DS stored in memory 104 so that I may be understood by this contractor.

[0022] As shown in step 302, the step of drawing 6 is started when a linkage editor 124 is started by -G option on a command line. The aforementioned -G option shows what a linkage editor 124 should generate a common (object which can be performed dynamically is contrastive) object for. In the gestalt of desirable implementation of this invention, said linkage editor 124 is started also by -M option. A **-M option shows what a mapfile 130 should be used for as the "version definition directions" source. Although the example of drawing 2 shows creation of a common object, this invention is usable also in order to create the relocatable object by which a version setup was carried out, and the object which can be performed dynamically.

[0023] The following examples do not include the detail of the Unix command (cat, cc, pvs, ld) used. This Unix command is indicated by available Solaris ReferenceManual from Sun Microsystems. A next paragraph explains an example of the common object source code 110 and a mapfile 130. Table 1 shows the source code 110 of four source files ("foo.c", "data.c", "bar1.c", and "bar2.c") written by the C programming language. These source code files constitute one common object. Moreover, the mapfile of Table 2 defines the global-area interface of various versions of said common object. These files constitute the relocatable object 112 of drawing 1 by being compiled. Then, said linkage editor 124 creates the common object 114 by which a version setup was carried out according to a mapfile 130 so that it may explain later.

[0024]

[Table 1]

```
$ cat foo.c

extern const char *  _fool;
extern const char *  _foo2;

void fool()
{
        (void) printf(_fool);
}

void foo2()
{
        (void) printf(_foo2);
}

$ cat data.c

const char * _foo1 = "string used by function foo1()\n";
const char * _foo2 = "string used by function foo2()\n";

$ cat bar1.c

extern void foo1();

void bar1()

{
        foo1();
}

$ cat bar2.c

extern void foo2();

void bar2()
{
        foo2();
}
```

[0025]
[Table 2]

```
$ cat mapfile
SUNW.1.1 {                          #Release X
        global:
                foo1;
        local:
                *;
};

SUNW.1.2 {                          #Release X+1
        global:
                foo2;
} SUNW.1.1;

SUNW.1.2.1 {  } SUNW.1.2;           #Release X+2

SUNW.1.3a {                         #Release X+3
        global:
                bar1;
} SUNW.1.2;

SUNW.1.3b {                         #Release X+3
        global:
                bar2;
} SUNW.1.2;

SUNW.1.4 {
        global:
                bar3;
} SUNW.1.3a  SUNW.1.3b
```

[0026] Drawing 8 shows a format of the mapfile 130 which used the Backus-Naur (BAKKASUNAURU) format. In this BAKKASUNAURU format, a bracket "[" and"]" show an optional element. For example, in drawing 8 , "[version name]" is the optional element of said mapfile format. The common object 114 offers the global symbol which can be together joined by other objects like the object 120 which can be performed dynamically at the time of activation. These global symbols are specified in said mapfile 130, and describe the application

binary interface (ABI) of the common object 114. The interface of an object can be changed by addition or removal of a global symbol during the life time of the common object 114. Furthermore, evolution of the common object 114 may be accompanied by modification on the internal activation to this object 114 which does not influence the global symbol of said interface.

[0027] Table 2 shows an example of a mapfile 130. In this table 2, said mapfile 130 includes SUNW.1.1 which is the version of the common object 114, SUNW.1.2, SUNW.1.2.1, SUNW.1.3a, SUNW.1.3b, and the version definition about SUNW.1.4. These version definitions contain all the versions (and those global symbols) by which this common object 114 was defined till then. In this example, SUNW.1.1 are the version of the common object 114 contained in Release X, SUNW.1.2 are the version of the common object 114 contained in Release X+1 which inherited the global symbol of SUNW.1.1, and SUNW.1.2.1 are the version of the common object 114 contained in Release X+2 which inherited the global symbol of SUNW.1.2. In addition, since SUNW.1.2.1 do not contain a new global symbol, they are a version "weak (weak: week)." This version shows modification when ["when / activation /"] it can set to said common object. Said other versions show modification of an "interface." Although SUNW.1.2.1 have functional modification of said common object, they show the version which does not have interface modification.

[0028] Furthermore, SUNW.1.3a is the version of the common object 114 contained in Release X+3 which inherited the global symbol of SUNW.1.2, and is the version of the common object 114 contained in Release X+3 from which SUNW.1.3b also inherited the global symbol of SUNW.1.2. However, the SUNW.1.3b version defines the global symbol "bar2" instead of the global symbol "bar1" defined as SUNW.1.3a. SUNW.1.4 define a global symbol "bar3" and have succeeded the global symbol of said SUNW.1.3a and SUNW.1.3b.

[0029] In the example of the table of Table 2, a notation "foo1" is the only global symbol defined as the public interface of SUNW.1.1 version. With a special "automatic contraction-ized" instruction ("*"), the global symbol of all others of said common object is reduced by the local scope so that it may not become some interfaces over this common object. In this way, the public interface of said SUNW.1.1 version is constituted by the internal version definition of the version relevant to said global symbol "foo1."

[0030]

[Table 3]

```
$ cc -o libfoo.so.1 -M mapfile -G foo.c bar1.c bar2.c data.c
$ ln -s libfoo.so.1 libfoo.so
$ pvs -dsv libfoo.so.1
        libfoo.so.1:
                _end;
                _GLOBAL_OFFSET_TABLE_;
                _DYNAMIC;
                _edata;
                _PROCEDURE_LINKAGE_TABLE_;
                _etext;
        SUNW.1.1:
                foo1;
                SUNW.1.1;
        SUNW.1.2:               {SUNW.1.1};
                foo2;
                SUNW.1.2;
        SUNW.1.2.1 [WEAK]:      {SUNW.1.2};
                SUNW.1.2.1;
        SUNW.1.3a:              {SUNW.1.2};
                bar1;
                SUNW.1.3a;
        SUNW.1.3b:              {SUNW.1.2};
                bar2;
                SUNW.1.3b;
        SUNW.1.4:               {SUNW.1.3a SUNW.1.3b};
                bar3;
                SUNW.1.4;
```

[0031] This table 3 shows the Unix command for compiling and linking the source code file of

Table 1. An object code file "foo.o", "data.o", "bar1.o", and "bar2.o" (not shown) are dynamically linked using the mapfile 130 of Table 2, and, thereby, generate the common object 114 called "libfoo.so.1." (In the case of this example, cc compiler calls the ld (1) linkage editor 124 automatically) . -G option on a command line shows what a linkage editor 124 should generate not the object that can be performed dynamically but a common object for. The "ln" command creates the "-l" "compile environment" name suitable for an option of ld (1). "pvs"Unix command prints out the version setting information on the common object created by said linkage editor 124, and a global symbol available to each version.

[0032] As shown in Table 3, the definition of a "criteria version" is also created about said object. This criteria version is defined using the identifier (for example, "libfoo.so.1") of said common object itself, and relates with said object the notation generated and secured by said linkage editor 124. For example, in the example of Table 3, the definition of a criteria version is created about common object libfoo.so.1 containing the global symbol (for example, _etext, _edata, _end, _DYNAMIC, PROCEDURE_LINKAGE_TABLE, and GLOBAL_OFFSET_TABLE) created and secured by said linker.

[0033] When return and a version (setting to step 304) name appear in a mapfile 130 at drawing 6 , said especially linkage editor 124 creates some sections about a version in the common object 114. These special sections are shown in drawing 9 , and include the version definitions section 506, the version notation section 508, and the optional version subordination condition section 510. The following paragraphs explain creation and use of these sections.

[0034] Drawing 10 shows the format of the version definitions section 506 of the common object 114 by which a version setup was carried out. Moreover, drawing 20 shows an example of the version definitions section based on Table 1 – 3. This version definitions section includes a section header 602 (refer to drawing 16), the structure section 604, a flag 606, the version definition index 610, counted value 612, a hash value 614, the auxiliary value 616, the version [degree] definition pointer 618, the identifier 620 of this version itself, and the identifier of the version from which the this defined version inherited the global symbol 620,622.

[0035] Each common object 114 contains one version definitions section 506 which has much version definitions. Each fields 604-622 are called "version definition." The following paragraph explains the contents of the version definition. Said section header 602 shows the number of the versions contained in said version definitions section. Each version definitions section includes the information about the criteria version which defines the global symbol which is not clearly defined as a mapfile. For this reason, the fields 604-622 (version definition) about a criteria version are created at step 306 of drawing 6 . The field 609 ("criteria" flag) is set up in this criteria version definition.

[0036] In step 308, a version definition (fields 604-622) is created by the linkage editor 124 for every version of the common object 114 defined as said mapfile 130. In this way, in step 308, six version definitions other than said criteria version definition are created about the mapfile 130 of Table 2. Steps 310-316 of drawing 7 are performed for each [which was created at step 306,308] the version definition of every. As shown in step 310, when said mapfile 130 does not define the global symbol about one version (for example, SUNW.1.2.1 reference of Table 1), the "week" flag 608 about the version definition is set up in step 312 (refer to drawing 20).

[0037] The field 604 is the version number of the structure itself. The version definition index 610 has the value by which the common object 114 was defined and from which it differs for every version (proper), and makes it explain with relation with drawing 11 later. Counted value 612 shows the number of a pair of instances of the fields 620 and 622 in the version concerned. A hash value 614 is a hash value to the identifier of this version, and uses the usual ELF (format which can be performed) Hash Function. Moreover, the auxiliary value 616 is an index to the 1st field 620 of this version. The version [degree] definition 618 is an index to the field 604 of a version definition of the degree in said version definitions section.

[0038] Step 313 creates the entry 620,622 to the identifier of this version, and it sets up said

auxiliary value 616 so that it may point to this entry. In this way, said 1st field 620 includes the version name of said version itself. Succession (inheriting) information consists of identifiers of one which becomes the origin from which the defined version inherits other versions, or two versions or more. In step 314 of drawing 7 , it investigates whether a mapfile 130 has succession information, as shown in drawing 8 and Table 2. In step 316, said linkage editor 124 creates one or two entries 620,622 or more holding succession information. This field 620,622 exists for every version which becomes the origin from which a specific version inherits other versions. Said field 622 points to the next field 620 (or zero) including the identifier of the version definition from which said field 620 was inherited.

[0039] In step 318 of drawing 6 , said linkage editor 124 creates a version notation section about all the notations of the object in a link edit. Drawing 11 is drawing showing a format of the version notation section 508. The entry in said version notation section 508 corresponds to the notation in the symbol table about the version, and 1 to 1 (refer to drawing 21). Said symbol table is well known by this contractor, and is indicated by said System V Application Binary Interface Manual, therefore is not explained here. A format of said section header 702 is explained in relation with drawing 17 . The entry 704 of said version notation section 508 is the index of a version with which the notation was defined. In Table 2, when a SUNW.1.3b version has the index 610 of "6" (refer to drawing 20), the entry of the version notation section corresponding to a global symbol "bar2" will have the entry value of "6" (drawing 21 R> 1). As shown in drawing 11 , the entry of a notation which has a local scope has the value of "0." The entry of the notation in a criteria version definition has the value of "1." Like ****, only the notation (for example, the notation about a criteria section, the identifier of each version, "foo1", "foo2", "bar1", "bar2", and "bar3") clearly defined as a global symbol has the entry 704 which is not zero in the version notation section 508. Step 320 of drawing 6 creates the version subordination condition section about the common object 114 (when required). For example, refer to other objects by which a version setup was carried out for the common object 114. Creation of this version subordination condition section is explained later.

[0040] b. The creation table 4 of the version subordination status information at the time of construction shows an example of the source version of the application program software 116 ("prog.c"). Refer to "two global symbols 1 of common object 114libfoo.so.1, i.e., "foo", and the foo2" for "prog.c." These notations are defined as a part of interface SUNW.1.1 and SUNW.1.2, respectively. According to Table 4, Compiler cc starts the ld linkage editor 124 at the time of construction. Association (binding) which contains a global symbol "foo1" and "foo2" between "prog.c", and version SUNW.1.1, SUNW.1.2 and SUNW.1.2.1 at the time of construction is performed. The two former versions show notation association. The latter version is recorded for the weak property. Since version control lead is not given by compile/link command, said linkage editor 124 checks all the versions of the common object 114 which exists when decomposing the global symbol in prog.c.

[0041]

[Table 4]

```
$ cat prog.c
extern void foo1();
extern void foo2();

main ()
{
        foo1();
        foo2();
}
$ cc -o prog prog.c -L. -R. -lfoo
$ pvs -r prog
        libfoo.so.1 (SUNW.1.2 SUNW.1.2.1);
```

[0042] Drawing 12 shows the step performed by said linkage editor 124 at the time of construction, in order to create the ELF file 120 which can be dynamically performed from the relocatable object 118 and the common object 114. The format of said ELF file 120 which can

be performed dynamically is shown in drawing 14 . Although a format of drawing 14 includes a version subordination condition section, it is the same as a format of drawing 9 except for the point of not including a version definitions section or a version notation section. Drawing 15 is drawing showing a format of a version subordination condition section.

[0043] in step 802 of drawing 12 , said linkage editor 124 refers for the global symbol which is not decomposed in the object under link (for example, prog) with the global-symbol table of other objects under link (for example, libfoo1.so.1) -- said object prog -- said -- others -- it judges whether it is subordinate to the object. When subordinate, in step 804, said linkage editor 124 judges whether whether the object needed having two or more versions and said object needed have a version definitions section. In step 806, when only some versions of said object needed are judged for said linkage editor 12 to be the thing of visibility, control progresses to step 810. When that is not right, control progresses to step 808.

[0044] In step 808, said linkage editor 124 creates a version subordination condition section to the object 120 which can be performed dynamically by investigating all the available version definitions sections of the common object 114, and judging which version contains a global symbol required for the relocatable object 118. Or a mapfile 132 may identify only some versions as a thing of visibility for said linkage editor 124. At step 810, it judges whether said linkage editor 124 investigates the version definitions section of the visibility of the common object 114, and contains a global symbol required for the object 118 with which relocatable version.

[0045] The subordination condition recorded on the version definitions section of drawing 14 is created whenever an application program refers to the global symbol of an interface to a common object. Table 4 shows an example of the source version of the application program software 116 ("prog.c"). Refer to the global symbol "foo2" (refer to Table 2) defined as the global symbol "foo1" and SUNW.1.2 version which were defined as SUNW.1.1 version for this application program "prog.c." In this way, a dependency exists between application program prog.c and SUNW(it inherited from SUNW.1.1 version).1.2 version. Said linkage editor 124 creates the version subordination condition section which shows that prog.c is subordinate to the object 120 which can perform prog.c dynamically at SUNW.1.2 version (refer to drawing 22). or [that to which said notation belongs to which version when said linkage editor 124 investigates the version definitions section and version notation section of said relocatable object 118 and all the common objects linked for every global symbol of the undefined in said relocatable object 118] -- about -- information is acquired.

[0046] Drawing 15 shows a format of the version definitions section 510. This version definitions section 510 contains the section header 1102 explained in relation to drawing 16 , the structure version 1104, counted value 1106, the file name 1108, the auxiliary value 1110, the version [degree] subordination condition section 1112, and the instance of two or more fields 1114-1122. said fields 1114-1122 -- a hash value 114 and a "week" -- a flag 1116, the intact field 1118, the identifier field 1120, and the identifier [degree] field 1122 are included.

[0047] As mentioned above in relation to drawing 10 , the structure version value 1104 is not related to a version setup of this invention. Counted value 1106 shows the number of the instances of the fields 1114-1122. A file name 1108 is the identifier of the subordination condition of a common object. The auxiliary value 1110 is an index to the 1st field 1114 about this version. The version [degree] subordination condition section value 1112 is an index (or zero) to the next version field 1104. Each instance of said fields 1114-1122 shows the version which an object in preparation needs. This version is needed when a version defines the global symbol referred to by said object. At least one instance of said fields 1114-1122 always exists.

[0048] A hash value 114 is generated from the version name 1120 using the usual ELF Hash Function. The "week" flag 1116 shows whether it is that in which it does not contain the global symbol that it is a version with the weak version [subordinate / (needed)], i.e., this version. The field 1118 is not formed in the gestalt of all implementation of this invention, and is included only for alignment. The 1st instance of the name field 1120 includes the identifier (for example, "SUNW.1.2") of said version itself. The identifier [degree] field 1122 is a

pointer in which the following hash value 1114 is shown.
[0049] "Version contraction-ization" to which all succession (inherited information) is made not to be recorded on the version definitions section 510 is used for the gestalt of this implementation of this invention. a weak version -- and the version needed is recorded. However, when these versions inherit a global symbol from other versions, generally a version besides the above is not recorded. For example, being contraction-ized is desirable before the succession through the 2nd version from the 1st weak version is recorded on the fields 1120 and 1122. Similarly, being contraction-ized is desirable before succession through the 2nd version from the 1st version which is not weak which is not weak is also recorded. Succession between a weak version and the version which is not weak is not contraction-ized. In this way, although SUNW.1.2 are recorded on the fields 1120 and 1122, succession from SUNW.1.1 is not contraction-ized. Furthermore, the subordination condition over SUNW.1.2.1 which are a weak version is recorded.
[0050] Drawing 16 - drawing 18 show drawing 10 , drawing 11 , and various modalities of the section of drawing 15 . Since the field of drawing 1616 - drawing 18 is a part of conventional ELF format, these all do not explain but explain only the field related to this invention here. Drawing 16 is drawing showing a format of the section header used for section headers 607, 702, and 1102. In the "sh_type" field 1204, the "sh_name" field 1202 includes the class of said section including the identifier of a section. Especially the value 1202 of said field has relation in this invention. Drawing 17 shows the value showing various classes of said field 1204. Especially the value 1302 has relation in this invention. The section header format of drawing 16 includes further the "sh_size" field 1306 which shows the byte count in said section, and the field 1308 and the "sh_link" "sh_info" field 1310. Drawing 18 shows the example value of the above "sh_type", "sh_link", and "sh_info." Especially the value 1402 has relation in this invention.
[0051] The above-mentioned paragraph explained creation of the ELF object 114 including the version definitions section 506 and the version notation section 508 at the time of construction, and creation of the ELF object 120 including the version subordination condition section 510. In addition, said version definitions section 506 and the version notation section 508 are created by only the object including the definition of a global symbol, and the version subordination condition section 510 is created by only the object subordinate to the object containing a global symbol. In this way, the common object which has a global-area interface will include sections 506,508 and 510. The application program which refers to said common object and which can be performed will include only the version subordination condition section 510.
[0052] c. The limit above-mentioned paragraph of the application structure at the time of construction explained an example of version association between all the available versions of the common object 114, and a relocatable object. Association can also be restricted so that it may generate only between an application program and the specific version of the object of drawing 13 . Drawing 13 shows a format of the file control directions in the mapfile 132 of drawing 4 . File control directions of this format are used in order to combine the object which can be performed dynamically according to the specific version of other objects.
[0053]
[Table 5]

```
$ cat mapfile
libfoo.so - SUNW.1.1;

$ cc -o prog prog.c -M mapfile -L. -R. -lfoo
Undefined                       first referenced
 symbol                             in file
foo2                            prog.o (symbol belongs to \
unavailable version ./libfoo.so (SUNW.1.2))
ld: fatal: Symbol referencing errors.  No output written to prog
```

[0054] As shown in drawing 4 and Table 5, when said linkage editor is performed using the mapfile 132 including the file control shown in drawing 13 , association is performed only about the specific version of said object. In Table 5, the mapfile contains "libfoo.so-SUNW.1.1." As

shown in Table 4, refer to a global symbol "foo1" and the "foo2" for application software "prog.c." The notation "foo2" is defined as SUNW.1.2 version. (Using cc compiler and ld (1) linkage editor) When prog.c is linked with -M option, this prog.c is linked only with SUNW.1.1 version. Thus, as shown in Table 5, a linkage editor 124 recognizes "foo2" defined as SUNW.1.2 version as a notation of the undefined.

[0055] d. The weak promotion table 6 of a version at the time of construction shows the example forced "so that a linkage editor 124 may promote a weak version (SUNW.1.2.1) to a strong version." With "-u" option, a linkage editor 124 records the subordination condition of "prog" to SUNW.1.2.1 version on the version subordination condition section of ** "prog." Furthermore, the "week" flag 1116 is set as a "false" in the version subordination condition section of "prog" about SUNW.1.2.1 version. In this case, SUNW.1.2.1 version inherits SUNW.1.2 version. Therefore, since all the subordination conditions of prog are "strength", contraction-ization of a version means what only SUNW.1.2.1 version is recorded for on prog as a version which is not weak. Promotion of such a weak version guarantees what it is checked for by the execution-time linkage editor 124 that the weak version exists till then, when "prog" is performed.

[0056]

[Table 6]

```
$ cc -o prog prog.c -L. -R. -u SUNW.1.2.1  -lfoo

$pvs - r prog
      libfoo.so.1 (SUNW.1.2.1)
```

[0057] 3. Check drawing 19 of the version setting information at the time of activation is the flow chart 1500 which shows the step performed by the execution-time linker 126 (refer to drawing 5), in order to guarantee that all the required versions of the object referred to exist, when linking and performing the object 120 which can be performed dynamically. The step of drawing 19 R> 9 is performed by executing preferably the instruction with which CPU102 was stored in memory. The step of drawing 19 R> 9 is performed as a front [activation] check, in order to judge whether all the required subordination conditions exist.

[0058] In step 1502, said linker 126 judges whether the object 120 which can perform under activation dynamically includes the version subordination condition section (refer to drawing 15). When this judgment result is NO, in any way, a check is not made but the usual activation is continued. When this judgment result is YES, in step 1504, said linker 126 judges whether at least one of the objects 114 under link includes the version definitions section and the version notation section. When this judgment result is NO, the check beyond it is not made but the usual activation is continued. When this judgment result is YES, control progresses to step 1506.

[0059] In step 1506, said execution-time linker 126 judges whether all the version subordination conditions in the current object 120 which can be performed dynamically were processed. When all the version subordination conditions are processed, the usual activation is continued, and when that is not right, control progresses to step 1508. At step 1508, said execution-time linker 126 judges whether the version needed (version subordination condition section of said object 120 which can be performed dynamically), and the version definitions section of said common object 114 are in agreement. For example, in drawing 20 and drawing 22 , SUNW.1.2 version and SUNW.1.2.1 version are defined as the version definitions section of the common object 114, and are specified in the version subordination condition section of the object 120 which can be performed dynamically if needed.

[0060] Since the version needed will not exist when the version for which the common object 114 is needed when coincidence is detected in step 1508 will exist and coincidence is not detected in step 1508, in step 1510, it is necessary to judge whether the version which does not exist is a weak version, or it is the version which is not weak. Said execution-time linker 126 judges whether it is a thing with said weak version by checking the "week" flag 1116 of the version subordination condition section of the object 120 which can be performed dynamically. (In addition, said flag 1116 may reflect the promotion from the weakness of a

version to strength) . When said execution-time linker 126 judges with it being a version with the weak version in step 1510, an error is not generated but control returns to step 1506. Since the version which is needed and which is not weak does not exist when [which is not a weak version] it judges, a fatal error occurs. Therefore, although subordination in a version does not generate an error, subordination in the "weak" version which does not exist and "which is not weak" will generate an error.

[0061] Drawing 20 shows an example of the version definitions section 1600, and drawing 21 shows an example of the version notation section 1700 about the common object 114 (libfoo.so.1 of Table 1 - 3). Furthermore, drawing 22 shows an example of the version subordination condition section 1800 about the gestalt of an application program 120 ("prog" of Table 4) which can be performed dynamically. (In this example, the common object 114 is subordinate to other objects, therefore does not have a version subordination condition section) .

[0062] Drawing 20 includes six version definitions for the criteria version definition 1604 and each version SUNW.1.1, SUNW.1.2, SUNW.1.2.1, SUNW.1.3a, SUNW.1.3b, and SUNW.1.4. The "criteria" flag of said criteria version definition 1604 is set up "truly [truly (true)]." Moreover, the "weak" flag of the version definition about SUNW.1.2.1 version is set up "truly [truly (true)]." Each version definition has the version definition index of a proper. Contraction-ization of a version is applied in a version definition table. Refer to a global symbol foo1 and the global symbol (SUNW.1.1) (SUNW.1.2) foo2 for application "prog." Since SUNW.1.2 inherit the global symbol of SUNW.1.1, said linkage editor 124 applies version contraction-ization, and an entry (SUNW.1.2 sake) is made in a version definitions section. SUNW.1.2.1 weak version is also recorded.

[0063] Drawing 21 is drawing showing some of the notations in the version notation section 1700. For example, the global variable "foo1" is defined as SUNW.1.1 version, and has a version definition index "2." In this way, the entry of said version notation section corresponding to the entry of the symbol table about foo1 contains "2." In the gestalt of the above-mentioned implementation, the identifier of each version (for example, SUNW.1.1) is also the global symbol created when generating the version definition.

[0064] Drawing 22 is drawing showing an example of the version subordination condition section in the object 120 which can be performed dynamically. By version contraction-ization, said section contains the entry for SUNW.1.2 instead of SUNW.1.1. This section also contains the entry for SUNW.1.2.1 "weak" version. A linker 126 judges with judging with prog having a version subordination condition section (step 1502), and libfoo.so.1 having a version definitions section at the time of activation (step 1504), and it judges with the version (SUNW.1.2) needed existing (step 1508). Therefore, "prog" will be performed. As mentioned above, although some desirable examples were explained per this invention, unless it deviates from the pneuma and the range not only of these examples but this invention, it will be understood that others' various deformation is possible.

[0065] It is as follows when some of invention concerning this application and its embodiment are summarized and shown in the last.

(1) The step which is the approach of giving version setting information to a software program, and prepares the 1st object code for the 1st software program, The step which prepares the mapfile which shows the version name about the version of said 1st software program, The information which shows the version name of said 1st software program so that it may be added to said 1st object code according to said mapfile The approach by which said 1st object code is linked, the step which generates by this the object by which a version setup was carried out is provided, and these steps are performed by data processing system.

(2) The step which prepares the 2nd object code for the 2nd software program, The step which links said 2nd object code to said object by which a version setup was carried out is provided further. The step said step to link judges the version of said 1st software program needed for said 2nd software program to be, An approach given in said 1st term characterized by including further the step which adds the information which shows said version needed for said 2nd software program to said 2nd object code, and generates by this the program which

can be performed dynamically.
(3) The approach of the publication by said 2nd term provide further the step which judges that the version needed in the program in which this activation is possible is in agreement with the version shown in said object by which a version setup was carried out before performing said program which can be performed dynamically, and the step perform said program which can perform dynamically, and said program which can perform dynamically calls said version of said object by which a version setup was carried out needed.
(4) An approach given in said 1st term containing the step which adds the information which shows the global symbol which constitutes said version name and the interface of said version according to said mapfile in order that the global symbol from which said mapfile constitutes the interface of said version of said 1st software program may be specified further and said step to generate may generate said object by which a version setup was carried out to said 1st object code.
(5) Said step to judge by investigating which global symbol is needed for said 2nd software program By judging the version of said 1st software program needed for said 2nd software program, and checking further the information in said object by which a version setup was carried out An approach given in said 2nd term containing the step which judges in which version said global symbol needed exists.
(6) An approach given in said 1st term whose information added to said 1st object code is a version definitions section.
(7) An approach given in said 1st term whose information added to said 1st object code is a version notation section.
(8) An approach given in said 2nd term whose information added to said 1st object code is a version subordination condition section.
(9) An approach given in said 1st term said whose object by which a version setup was carried out is a relocatable object.
(10) An approach given in said 1st term said whose object by which a version setup was carried out is an object which can be performed dynamically.
(11) An approach given in said 1st term said whose object by which a version setup was carried out is a common object.
[0066] (12) The storage which is equipment which gives version setting information to a software program, and stores the 1st object code for the 1st software program, The storage which stores the mapfile which specifies the version name about the version of said 1st software program, Equipment which attaches the additional information which defines the version name of said version of said 1st software program to said 1st object code according to said mapfile, and possesses by this the linker which generates the object by which a version setup was carried out.
(13) Equipment given in said 12th term which possesses further the linker which links said 2nd object code to said object by which a version setup was carried out, and generates by this the program which can be performed dynamically by giving the additional information which shows the storage which stores the 2nd object code for the 2nd software program, and the version needed for said 2nd software program to said 2nd object code.
(14) Equipment given in said 13th term which possesses further the execution-time linker which makes possible said program execution which can be performed dynamically when said version which was specified in said object which can be performed dynamically, and which is needed judges with it being in agreement with the version defined as said object by which a version setup was carried out.
(15) Equipment given in said 12th term said whose object by which a version setup was carried out is a relocatable object.
(16) Equipment given in said 12th term said whose object by which a version setup was carried out is an object which can be performed dynamically.
(17) Equipment given in said 12th term said whose object by which a version setup was carried out is a common object.
[0067] (18) In order to make it judge that the version of the object needed for the object

which can be performed dynamically exists during activation of this executable object, It is the computer program product equipped with the usable medium by computer which stored the code which can be read by computer. The 1st program code equipment which makes the 1st object code for the 1st software program prepare for a computer and which can be computer read, The 2nd program code equipment the mapfile which specifies the version name relevant to the version of said 1st software program as a computer is made to prepare for and which can be computer read, According to said mapfile, the information which shows the version name of said 1st software program so that it may be added to said 1st object code The computer program product which was made to link said 1st object code to a computer, and possesses by this the 3rd program code equipment which makes the object by which a version setup was carried out generate, and which can be computer read.
(19) The 4th program code equipment which makes the 2nd object code for the 2nd software program prepare for a computer and which can be computer read, The version of said 1st software program needed for a computer at said 2nd software program is made to judge. By making the information which shows the version needed for said 2nd software program add to said 2nd object code A computer program product given in said 18th term which possesses further the 5th program code equipment which performs the link of said 2nd object code to said object by which a version setup was carried out, and which can be computer read.
(20) A computer program product given in said 19th term characterized by to provide further the 6th program code equipment which makes it judge to be in agreement with the version defined as the object in which the version needed for a computer in order that said program which can be executed dynamically may execute a program has said version, and which can be computer read, and for said program which can execute dynamically to call said version of said object by which a version setup was carried out needed.
[0068]
[Effect of the Invention] As mentioned above, since this invention enables the upgrade of only a part required whenever the new version of an object is created, in order to abolish the need of changing the identifier of this object and to cope with version modification, it does so the outstanding effectiveness that an easy and efficient version setup is realizable.

[Translation done.]

* NOTICES *

1.This document has been translated by computer. So the translation may not reflect the original precisely.
2.**** shows the word which can not be translated.
3.In the drawings, any words are not translated.

DESCRIPTION OF DRAWINGS

[Brief Description of the Drawings]
[Drawing 1] The block diagram showing the computer system concerning this invention.
[Drawing 2] Drawing showing I/O of the linkage editor of drawing 1 at the time of construction.
[Drawing 3] Drawing showing other I/O of the linkage editor of drawing 1 at the time of construction.
[Drawing 4] Drawing showing other I/O of the linkage editor of drawing 1 at the time of construction.
[Drawing 5] Drawing showing I/O of the execution-time linker program of drawing 1 at the time of activation.
[Drawing 6] The flow chart which shows the step performed by the linkage editor of drawing 2 in order to add a version definitions section and a version notation section to an object.
[Drawing 7] The flow chart which shows the detail of processing of drawing 6 .
[Drawing 8] Drawing showing a format of the mapfile of drawing 2 .
[Drawing 9] Drawing showing the output of the linkage editor of drawing 2 contained in the object by which a version setup was carried out.
[Drawing 10] Drawing showing a format of the version definitions section of drawing 9 .
[Drawing 11] Drawing showing a format of the version notation section of drawing 9 .
[Drawing 12] The flow chart which shows the step performed by the linkage editor of drawing 3 or drawing 4 in order to add a version subordination condition section to the application program which can be performed dynamically.
[Drawing 13] Drawing showing a format of the mapfile of drawing 4 .
[Drawing 14] Drawing showing the output of the linkage editor of drawing 3 contained in the application program which can be performed dynamically.
[Drawing 15] Drawing showing a format of drawing 9 and the version subordination condition section of drawing 14 .
[Drawing 16] Drawing showing a format of the section header of drawing 10 , drawing 11 , and drawing 15 .
[Drawing 17] Drawing showing the list of various values in the header of drawing 16 .
[Drawing 18] Drawing showing the list of various values in the header of drawing 16 .
[Drawing 19] The flow chart which shows the step performed by the execution-time linker of drawing 5 in order to check that it is in agreement with the version to which the requirements for a version of an application program exist in the object linked to this application program.
[Drawing 20] Drawing showing an example of the version definitions section added to the object by said linkage editor.
[Drawing 21] Drawing showing an example of the version notation section added to the object by said linkage editor.
[Drawing 22] Drawing showing an example of the version subordination condition section which can be added to both the object in which a version setup was carried out by said linkage editor, and both [one side or] which can be performed dynamically.
[Description of Notations]
102 CPU
104 Memory

110 Source Code
114 Common Object
116 Source Code
118 Relocatable Object
120 Object Which Can be Performed Dynamically
124 Linkage Editor
124 Execution-Time Linker
130 Mapfile
132 Mapfile

[Translation done.]